RoomOS / Collaboration Endpoint software
AUGUST 2019

# xAPI over WebSocket

for products running RoomOS or CE software

Thank you for choosing Cisco!

Your Cisco product has been designed to give you many years of safe, reliable operation.

This part of the product documentation is aimed at integrators working with the xAPI (Application Programming Interface) of the device.

Our main objective with this guide is to address your goals and needs. Please let us know how well we succeeded!

May we recommend that you visit the Cisco web site regularly for updated versions of this guide.

The user documentation can be found on:

- https://www.cisco.com/go/sx-docs
- https://www.cisco.com/go/mx-docs
- https://www.cisco.com/go/dx-docs
- https://www.cisco.com/go/room-docs
- https://www.cisco.com/go/board-docs

## How to use this guide

The top menu bar and the entries in the Table of contents are all hyperlinks. You can click on them to go to the topic.

# Table of contents

**Introduction** ..... **3**
User documentation and software ..... 4
About the xAPI and WebSocket ..... 5

**WebSocket** ..... **6**
Enabling WebSocket on the device ..... 7
Setting up the WebSocket connection ..... 8

**JSON-RPC objects** ..... **9**
JSON-RPC objects for xAPI commands ..... 10
General description of the JSON-RPC objects ..... 11
Get a single leaf node ..... 12
Get a subtree ..... 13
Get a single leaf node from inside an array ..... 14
Query with wildcard in path ..... 15
Get complete status or configuration documents ..... 16
Set a configuration ..... 17
Issue a command ..... 18
Issue a command with multiple instances of the same argument ..... 19
Issue a multi-line command ..... 20
Register feedback ..... 21
Deregister feedback ..... 22
Receiving feedback ..... 23
Error codes ..... 24

Chapter 1

# Introduction

# User documentation and software

## Products covered in this guide

- Cisco Webex Board 55/55S
- Cisco Webex Board 70/70S
- Cisco Webex Board 85S
- Cisco Webex Room Kit Mini
- Cisco Webex Room Kit
- Cisco Webex Room 55
- Cisco Webex Room 55 Dual
- Cisco Webex Room 70 Single
- Cisco Webex Room 70 Dual
- Cisco Webex Room 70 Single G2
- Cisco Webex Room 70 Dual G2
- Cisco Webex Codec Plus [1]
- Cisco Webex Codec Pro [2]
- Cisco TelePresence DX70
- Cisco TelePresence DX80
- Cisco TelePresence SX10 Quick Set
- Cisco TelePresence SX20 Quick Set
- Cisco TelePresence SX80
- Cisco TelePresence MX200 G2
- Cisco TelePresence MX300 G2
- Cisco TelePresence MX700
- Cisco TelePresence MX800
- Cisco TelePresence MX800 Dual

## User documentation

This guide is an ammendment to the API Reference guide of the product. This guide provides you with the information required to connect to and use the API over WebSocket.

The guide applies both to on-premise registered devices (CUCM, VCS) and devices that are registered to our cloud service (Cisco Webex).

### Documentation on the Cisco web site

Visit the Cisco web site regularly for updated versions of the guides:

- ▶ https://www.cisco.com/go/sx-docs
- ▶ https://www.cisco.com/go/mx-docs
- ▶ https://www.cisco.com/go/dx-docs
- ▶ https://www.cisco.com/go/room-docs
- ▶ https://www.cisco.com/go/board-docs

### Documentation for cloud registered devices

For more information about devices that are registered to the Cisco Webex cloud service, visit the Help Center:

- ▶ https://help.webex.com

## CE software (on-premise)

Supported on CE9.7 and later.

Download software for the device from the Cisco web site:

- ▶ https://software.cisco.com/download/home

We recommend reading the Software release notes:

- ▶ https://www.cisco.com/c/en/us/support/collaboration-endpoints/spark-room-kit-series/tsd-products-support-series-home.html

## RoomOS software (cloud)

Software is automatically installed on your device.

We recommend reading the *What's New* and *Known and Resolved Issues* articles for the devices regularly.

- ▶ https://help.webex.com/article/6ger7db
- ▶ https://help.webex.com/article/llygcp

[1] Included in Room Kit Plus and Room Kit Plus Precision 60

[2] Included in Room Kit Pro and Room Kit Pro Precision 60

# About the xAPI and WebSocket

There are several ways to access the xAPI of the device:

- SSH, Telnet[1], HTTP/HTTPS, Serial connection[2]
- Macros
- WebSocket

Integrators should choose the connection method that suits their application the best. Regardless of which method you choose, the structure of the xAPI is the same.

This document is about how to use *WebSocket* to interact with the xAPI of the devices. You still need the API Reference Guide for a complete description of the xAPI itself.

Find the most recent *API Reference Guide* here:

▶ https://www.cisco.com/c/en/us/support/collaboration-endpoints/spark-room-kit-series/products-command-reference-list.html

## xAPI over WebSocket

There are several benfits to using WebSocket. The most prominent ones are:

- The communication channel over a WebSocket is open both ways until it is explicilty closed. This means that the server can send data to the client as soon as the new data is available, and there is no need for reauthentication for every request. This improves speed significantly compared to HTTP.
- Each message contains a complete JSON document and nothing else. There is no need to parse text, or a mix of text and XML.
- Many porgramming languages has good library support for WebSocket and JSON-RPC.

## Software support

All devices that run the following software versions support xAPI over WebSocket:

- CE9.7 or later
- RoomOS (when available)

## Terminology

### Devices or systems?

For cloud registered devices, we refer to our products as Boards, Room and Desk devices. For on-premise the same products are also referred to as codecs, endpoints, video devices, video systems, or simply systems.

In this document we use the term *device*.

### xAPI or API?

The native API of our devices are most often referred to just as the API of the device.

In this document we use the term *xAPI*.

[1] Telnet is only available for DX, MX, and SX series.

[2] Serial connection is not available for DX70, DX80, Room 55 Dual, and Room 70.

Chapter 2

# WebSocket

# Enabling WebSocket on the device

WebSocket is not enabled by default. You can enable it using the web interface of the device or by issuing API commands.

Note that WebSocket is tied to HTTP, so that also HTTP or HTTPS must be enabled before you can use WebSocket.

## Using the Web interface

1. Sign in to the web interface of the device and navigate to *Setup > Configuration*
2. Set *NetworkServices > HTTP > Mode* to either **HTTP+HTTPS** or **HTTPS**.
3. Set *NetworkServices > Websocket* to **FollowHTTPService**.

## Using API commands

1. Sign in to an API command line interface for the device.
2. Issue one of these commands to enable HTTP or HTTPS:

   ```
   xConfiguration NetworkServices HTTP Mode: HTTP+HTTPS
   xConfiguration NetworkServices HTTP Mode: HTTPS
   ```

3. Issue this command to enable WebSocket:

   ```
   xConfiguration NetworkServices Websocket: FollowHTTPService
   ```

# Setting up the WebSocket connection

First you have to set up an HTTP/HTTPS connection to the device. Then you upgrade this connection to a WebSocket (using the HTTP Upgrade header field). When that's done, you can start exchanging xAPI messages in JSON-RPC 2.0 objects over the WebSocket.

The device URL is:

- `ws://<ip-address of the device>/ws` (unencrypted)
- `wss://<ip-address of the device>/ws` (encrypted)

## Authentication

We support the following authentication mechanisms when setting up the WebSocket connection. Both are using HTTP header fields:

A. *Basic authentication*

The user must provide a valid username/password combination using basic access authentication before the HTTP connection is upgraded to a WebSocket. Basic autentication uses the `Authorization` HTTP header field.

Syntax:

```
Authorization: Basic <credentials>
```

where `<credentials>` is the base64 encoding of `username:password`

B. *Authentication using the auth protocol header* *

The user must authenticate using an *auth protocol header*, as defined by Cisco. It builds on the `Sec-WebSocket-Protocol` HTTP header field. This method is required for browser based clients, which have no direct control over the `Authorization` header.

The authentication is sent as a URL-friendly base64 encoded string of `username:password`, as specified in RFC 4648 section 5.

Syntax:

```
Sec-WebSocket-Protocol: auth-<credentials>
```

where `<credentials>` is the base64 encoding of `username:password`

*JavaScript example:*

```
var ws = new WebSocket('wss://codec.example.com/ws',
'auth-' + btoa('user:password').replace(/[\/+=]/g,
function(c){return {'+':'-','/':'_','=':''}[c]}));
```

The base64 encoded username and password must be URL-friendly, hence the character replace function.

## About WebSocket

The WebSocket protocol is specified in RFC 6455.

For general information about WebSocket, go to:

► https://en.wikipedia.org/wiki/WebSocket

* Supported in RoomOS, and CE9.8 and later.

Chapter 3

# JSON-RPC objects

# JSON-RPC objects for xAPI commands

On the next pages you will first find a general description of the JSON-RPC objects that we use to execute xAPI commands:

- Request (from client to server)
- Response (from server to client)
- Notification (from server to client)

The objects are based on the JSON-RPC 2.0 (Remote Procedure Call) protocol. Refer to:

► https://www.jsonrpc.org/specification

Then, there are examples of how to use the objects for the following types of xAPI actions.

## Get status and configuration values

- Get a single status or configuraton value (a single leaf node)
- Get a group of statuses or a group of configurations (a subtree)
- Get a single status or configuration value from inside an array (a node from inside an array)
- Query for multiple statuses or configurations using wildcards
- Get the complete document of statuses or configurations

## Issue commands and set configurations

- Issue a single-line commands
- Issue a multi-line command
- Set a configuration

## Handling feedback

- Register for feedback
- Deregister feedback
- Receiving feedback

# General description of the JSON-RPC objects

A **Request** object has the following structure:

```
{
   "jsonrpc": "2.0",
   "id": <id>,
   "method": <method>,
   "params": <Parameters>
}
```

- `"id": <id>,` — An identifier of this request. The server must reply with the same value in the Response object. If `"id"` is not included, it is a notification, which means that the server doesn't send a response.
- `"method": <method>,` — A String containing the name of the method to be invoked. For example `"xGet"`, `"xQuery"`, or `"xSet"`.
- `"params": <Parameters>` — An Object that holds the parameter values to be used during the invocation of the method. The Object must have member names that match the names that the server expects.

A **Response** object has the following structure:

```
{
   "jsonrpc": "2.0",
   "id": <id>,
   "result": <result>,
   "error": <error>
}
```

- `"id": <id>,` — Required. It is the same as the value of the `"id"` in the Request object. Set to `NULL` if there was an error in detecting the `"id"` of the Request object.
- `"result": <result>,` — Required on success. The value is determined by the server. Do *not* include `"result"` if there was an error invoking the method.
- `"error": <error>` — Required on error. The value is an Object with the followoing members:
  - code (integer that indicates the error type)
  - message (short description of the error)
  - data (additional information about the error, may be omitted)

  Do *not* include `"error"` if there was no error triggered during invocation.

A **Notification** object has the following structure

```
{
   "jsonrpc": "2.0",
   "method": <method>,
   "params": <Parameters>
}
```

- `"method": <method>,` — A String containing the name of the method to be invoked. For example `"xFeedback/Event"`.
- `"params": <Parameters>` — An Object that holds the parameter values to be used during the invocation of the method. The Object must have member names that match the names that the client expects.

xAPI action: Get status and configuration values

# Get a single leaf node

Applies to both status and configurations. Use either "Status" or "Configuration" as the first element in the "Path" array.

Request:

```
{
   "jsonrpc": "2.0",
   "id": 101,
   "method": "xGet",
   "params": {
      "Path": ["Status", "SystemUnit", "Uptime"]
   }
}
```

Response:

```
{
   "jsonrpc": "2.0",
   "id": 101,
   "result": 62974
}
```

Request:

```
{
   "jsonrpc": "2.0",
   "id": 102,
   "method": "xGet",
   "params": {
      "Path": ["Configuration", "SystemUnit", "Name"]
   }
}
```

Response:

```
{
   "jsonrpc": "2.0",
   "id": 102,
   "result": "my-device"
}
```

## Corresponding commands and responses in terminal mode

Command:

```
xStatus SystemUnit Uptime
```

Response:

```
*s SystemUnit Uptime: 62974
** end

OK
```

Command:

```
xConfiguration SystemUnit Name
```

Response:

```
*c xConfiguration SystemUnit Name: "my-device"
** end

OK
```

xAPI action: Get status and configuration values

# Get a subtree

Applies to both status and configurations. Use either "Status" or "Configuration" as the first element in the "Path" array.

Request:

```
{
   "jsonrpc": "2.0",
   "id": 103,
   "method": "xGet",
   "params": {
      "Path": ["Status", "SystemUnit", "State"]
   }
}
```

Response:

```
{
   "jsonrpc": "2.0",
   "id": 103,
   "result": {
      "NumberOfActiveCalls": 0,
      "NumberOfInProgressCalls": 0,
      "NumberOfSuspendedCalls": 0
   }
}
```

## Corresponding commands and responses in terminal mode

Command:

```
xStatus SystemUnit State
```

Response:

```
*s SystemUnit State NumberOfActiveCalls: 0
*s SystemUnit State NumberOfInProgressCalls: 0
*s SystemUnit State NumberOfSuspendedCalls: 0
** end

OK
```

xAPI action: Get status and configuration values

# Get a single leaf node from inside an array

Applies to both status and configurations. Use either "Status" or "Configuration" as the first element in the "Path" array.

In order to retrieve a node from inside an array, you must add the index number in the path.

Request:

```
{
   "jsonrpc": "2.0",
   "id": 104,
   "method": "xGet",
   "params": {
      "Path": ["Status", "SystemUnit", "Hardware", "Monitoring", "Fan", 2, "Status"]
   }
}
```

Response:

```
{
   "jsonrpc": "2.0",
   "id": 104,
   "result": "2372 rpm"
}
```

## Corresponding commands and responses in terminal mode

Command:

```
xStatus SystemUnit Hardware Monitoring Fan 2
   Status
```

Response:

```
*s SystemUnit Hardware Monitoring Fan 2 Status:
   "2372 rpm"
** end

OK
```

xAPI action: Get status and configuration values

# Query with wildcard in path

Applies to both status and configurations. Use either "Status" or "Configuration" as the first element in the "Query" array.

"**" matches zero or more levels in the path.

Request:

```
{
   "jsonrpc": "2.0",
   "id": 105,
   "method": "xQuery",
   "params": {
      "Query": ["Status", "**", "DisplayName"]
   }
}
```

Response:

```
{
   "jsonrpc": "2.0",
   "id": 105,
   "result": {
      "Status": {
         "SIP": {
            "CallForward": {
               "DisplayName": "my-name"
            }
         },
         "SystemUnit": {
            "Software": {
               "DisplayName": "ce 9.7.1 ..."
            }
         }
      }
   }
}
```

Note the difference between xQuery and xGet:

- The response to xQuery always starts from the top node, i.e. "Status" or "Configuration".
- The response to xGet starts relative to the path given in the "Query".

## Corresponding commands and responses in terminal mode

Command:

```
xStatus // DisplayName
```

Response:

```
*s SIP CallForward DisplayName: "my-name"
*s SystemUnit Software DisplayName: "ce 9.7.1 ..."
** end

OK
```

xAPI action: Get status and configuration values

# Get complete status or configuration documents

Both xGet and xQuery can get an entire tree or a subtree. The trees or subtrees must be fetched separately by adding ["Configuration"] or ["Status"] in the "Path". ["**"] is not valid.

The response to xQuery always starts from the top node, i.e. "Status" or "Configuration". The response to xGet starts relative to the path given in the "Query", so "Status" or "Configuration" are never included.

In order to receive the complete status tree, issue one of these commands:

### Using xGet:

```
{
   "jsonrpc": "2.0",
   "id": 106,
   "method": "xGet",
   "params": {
      "Path": ["Status"]
   }
}
```

### Using xQuery:

```
{
   "jsonrpc": "2.0",
   "id": 107,
   "method": "xQuery",
   "params": {
      "Query": ["Status"]
   }
}
```

In order to receive the complete configuration tree, issue one of these commands:

### Using xGet:

```
{
   "jsonrpc": "2.0",
   "id": 108,
   "method": "xGet",
   "params": {
      "Path": ["Configuration"]
   }
}
```

### Using xQuery:

```
{
   "jsonrpc": "2.0",
   "id": 109,
   "method": "xQuery",
   "params": {
      "Query": ["Configuration"]
   }
}
```

## Corresponding commands and responses in terminal mode

### Command:

```
xStatus
```

### Response:

The complete status tree.

```
*s Audio Input Connectors HDMI 1 Mute: Off
*s Audio Input Connectors HDMI 2 Mute: Off
...
*s Video Selfview OnMonitorRole: First
*s Video Selfview PIPPosition: CenterRight
** end

OK
```

### Command:

```
xConfiguration
```

### Response:

Tthe complete configuration tree.

```
*c xConfiguration Audio DefaultVolume: 70
*c xConfiguration Audio Input ARC 1 Mode: On
...
*c xConfiguration Video Selfview OnCall
   Duration: 10
*c xConfiguration Video Selfview OnCall Mode:
   Off
** end
```

xAPI action: Issue commands and set configurations

# Set a configuration

Request:

```
{
   "jsonrpc": "2.0",
   "id": 110,
   "method": "xSet",
   "params": {
      "Path": ["Configuration", "SystemUnit", "Name"],
      "Value": "my-device"
   }
}
```

Response:

```
{
   "jsonrpc": "2.0",
   "id": 110,
   "result": true
}
```

## Corresponding commands and responses in terminal mode

Command:

```
xConfiguration SystemUnit Name: "my-device"
```

Response:

```
** end

OK
```

xAPI action: Issue commands and set configurations

# Issue a command

Request:

```
{
   "jsonrpc": "2.0",
   "id": 111,
   "method": "xCommand/Dial",
   "params": {
      "Number": "alice@example.com",
      "Protocol": "H320"
   }
}
```

Response:

```
{
   "jsonrpc": "2.0",
   "id": 111,
   "result": {
      "CallId": 2,
      "ConferenceId": 1
   }
}
```

Error response:

```
{
   "jsonrpc": "2.0",
   "id": 111,
   "error": {
      "code": 1,
      "data": {
         "Cause": 21
      },
      "message": "Not paired with isdn link"
   }
}
```

## Corresponding commands and responses in terminal mode

Command:

```
xCommand Dial Number: "alice@example.com"
   Protocol: H320
```

Response:

```
OK
*r DialResult (status=OK):
*r DialResult CallId: 2
*r DialResult ConferenceId: 1
** end
```

xAPI action: Issue commands and set configurations

# Issue a command with multiple instances of the same argument

Request:

```
{
   "jsonrpc": "2.0",
   "id": 115,
   "method": "xCommand/Presentation/Start",
   "params": {
      "Layout": "Equal",
      "ConnectorId": [2, 3],
      "SendingMode": "LocalRemote"
   }
}
```

Response:

```
{
   "jsonrpc": "2.0",
   "id": 115,
   "result": {
      "Instance": 1
   }
}
```

## Corresponding commands and responses in terminal mode

Command:

```
xCommand Presentation Start Layout: Equal
   ConnectorId: 2 ConnectorId: 3 SendingMode:
   LocalRemote
```

Response:

```
OK
*r PresentationStartResult (status=OK):
*r PresentationStartResult Instance: 1
** end
```

xAPI action: Issue commands and set configurations

# Issue a multi-line command

Multi-line commands are similar to regular commands. In addition to regular parmeters (if any), a multi-line command has a "body" parameter. The "body" parameter contains the paylod of the command, including line breaks.

Request:

```
{
   "jsonrpc": "2.0",
   "id": 112,
   "method": "xCommand/SystemUnit/SignInBanner/Set",
   "params": {
      "body": "You need admin credentials to continue.
Contact the IT Help Desk for more information.",
   }
}
```

Response:

```
{
   "jsonrpc": "2.0",
   "id": 112,
   "result": {
      "status": "OK"
      }
}
```

## Corresponding commands and responses in terminal mode

Command:

```
xCommand SystemUnit SignInBanner Set↵
You need admin credentials to continue.↵
Contact the IT Help Desk for more information.↵
.↵
```

Response:

```
OK
*r SignInBannerSetResult (status=OK):
** end
```

xAPI action: Handling feedback

# Register feedback

A device may give very much feedback, especially when calls are connected and disconnected. We recommend you to subscribe only to the feedback you need.

Request:

```
{
   "jsonrpc": "2.0",
   "id": 113,
   "method": "xFeedback/Subscribe",
   "params": {
      "Query": ["Status", "Video", "Selfview"],
      "NotifyCurrentValue": true
   }
}
```

When "NotifyCurrentValue" is set to true, you receive a feedback notification with the current value of the configuration or status nodes that match the Query.

Response:

```
{
   "jsonrpc": "2.0",
   "id": 113,
   "result": {
      "Id": 1
   }
}
```

This is the Id of the feedback registration.

You will find the same Id in the freedback notifications that you receive. And you must use the same Id when you want to unsubscribe to these feedback notifications (deregister).

Notification:

```
{
   "jsonrpc": "2.0",
   "method": "xFeedback/Event",
   "params": {
      "Id": 1,
      "Status": {
         "Video": {
            "Selfview": {
               "FullscreenMode": "Off",
               "Mode": "Off",
               "OnMonitorRole": "First",
               "PIPPosition": "CenterRight"
            }
         }
      }
   }
}
```

## Corresponding commands and responses in terminal mode

Command:

```
xFeedback Register Status/Video/Selfview
```

Response:

```
** end

OK
```

xAPI action: Handling feedback

# Deregister feedback

Request:

```
{
   "jsonrpc": "2.0",
   "id": 114,
   "method": "xFeedback/Unsubscribe",
   "params": {
      "Id": 1
   }
}
```

This is the Id of the feedback registration.

It was returned in the "result" object of the response to the feedback registration.

Response:

```
{
   "jsonrpc": "2.0",
   "id": 114,
   "result": true
}
```

Error response:

```
{
   "jsonrpc": "2.0",
   "id": 114,
   "error": {
      "code": -32602,
      "message": "Unknown subscription Id",
      "data": {
         "Id": 4
      }
   }
}
```

A feedback registration Id that doesn't exist.

## Corresponding commands and responses in terminal mode

Command:

```
xFeedback Deregister Status/Video/Selfview
```

Response:

```
** end

OK
```

xAPI action: Handling feedback

# Receiving feedback

Feedback is sent as JSON-RPC notifications. Notifications don't have an "id" attribute.

Notification:

```
{
   "jsonrpc": "2.0",
   "method": "xFeedback/Event",
   "params": {
      "Id": 1,
      "Status": {
         "Video": {
            "Selfview": {
               "Mode": "On"
            }
         }
      }
   }
}
```

This is the Id of the feedback registration.

It was returned in the "result" object of the response to the feedback registration.

```
{
   "jsonrpc": "2.0",
   "method": "xFeedback/Event",
   "params": {
      "Id": 1,
      "Status": {
         "Video": {
            "Selfview": {
               "FullscreenMode": "On"
            }
         }
      }
   }
}
```

## Corresponding feedback in terminal mode

Feedback:

```
*s Video Selfview Mode: On
** end
*s Video Selfview FullscreenMode: On
** end
```

# Error codes

## Error codes from JSON-RPC

-32600: Invalid Request
-32601: Method Not Found
-32602: Invalid Params
-32603: Internal Error
-32700: Parse Error

## Application-defined server errors

1: Command Error
-31999: Permission Denied
-31998: Subscriber Count Exceeded
-31997: Not Ready

# Cisco contacts

On our web site you will find an overview of the worldwide Cisco contacts.

Go to: ► https://www.cisco.com/go/offices

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Dr.
San Jose, CA 95134 USA

## Intellectual property rights